Cuda vs. MLK running FFT on 1GB of data

Presented by Stephen MacKenzie
Former VC++ member for 20 years.

Overview

- Purpose of the demo
- • Goals: high-throughput FFT, telemetry decoupling, GPU autotuning
- • Tools used: Python, CUDA, MKL, memory-mapped files, NVIDIA Perf
- This demo is running on an Intel Ultra 7 system with 32 GB of DDR5 RAM, NVMe SSD storage, and an NVIDIA GeForce RTX 4060 GPU. The GPU is using NVIDIA Studio drivers and is set to autotune mode for optimal performance.

Voltage Collection Pipeline

- - `packet_forwarder.py`: captures controller voltage packets
- - Memory-mapped file strategy: 8 parallel maps for immediate use
- - Benefits: zero-copy access (cpu side), scalable ingestion
- NOTE: I have included two zip files memmaps1.zip and memmaps2.zip, extract them into one directory and make sure the memmap_dir in main points to it.

FFT Reference Section

- Voltage Over Time = Composite Signal
- Fast Fourier Transform
- Fast Fourier Transform (diagram of the real data in this project)
- Simulation (diagram of python generated graph)
- Bibliography

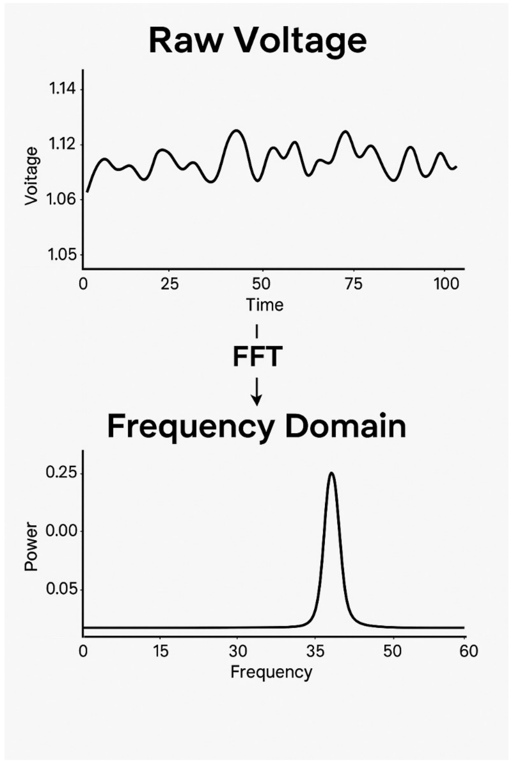
Voltage Over Time = Composite Signal

- When you sample voltage over time (e.g., V[0], V[1], ..., V[N]), you're capturing a single waveform. But that waveform is usually not a pure sine wave — it's a complex shape made up of many overlapping oscillations.
- 🧩Signals as Sums of Sine Waves
- Any periodic signal — even jagged or noisy ones — can be decomposed into a sum of sine and cosine waves at different frequencies, amplitudes, and phases. This is the essence of Fourier analysis.
- Your voltage array might look messy or smooth.
- But mathematically, it's just a weighted combination of frequencies.
-

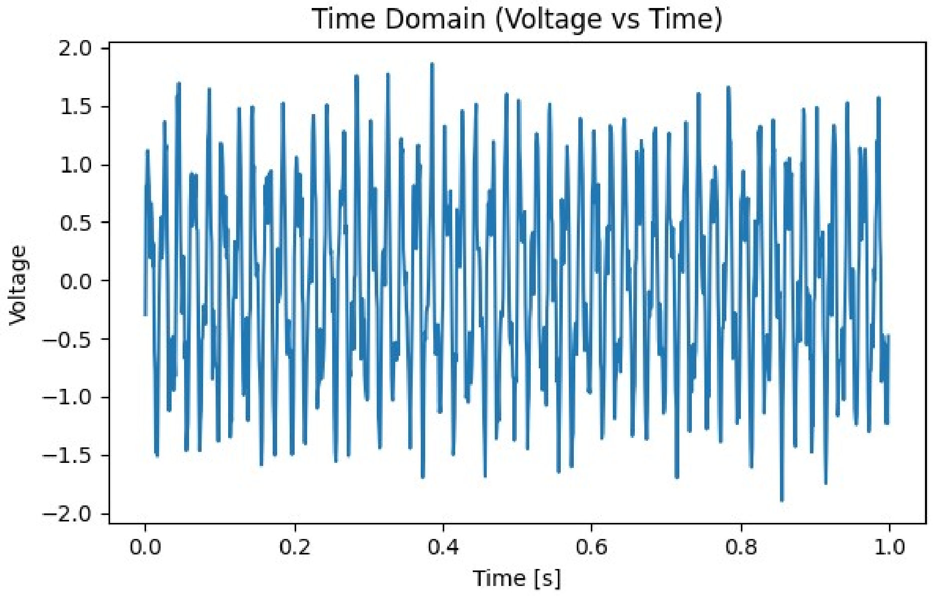
Fast Fourier Transform

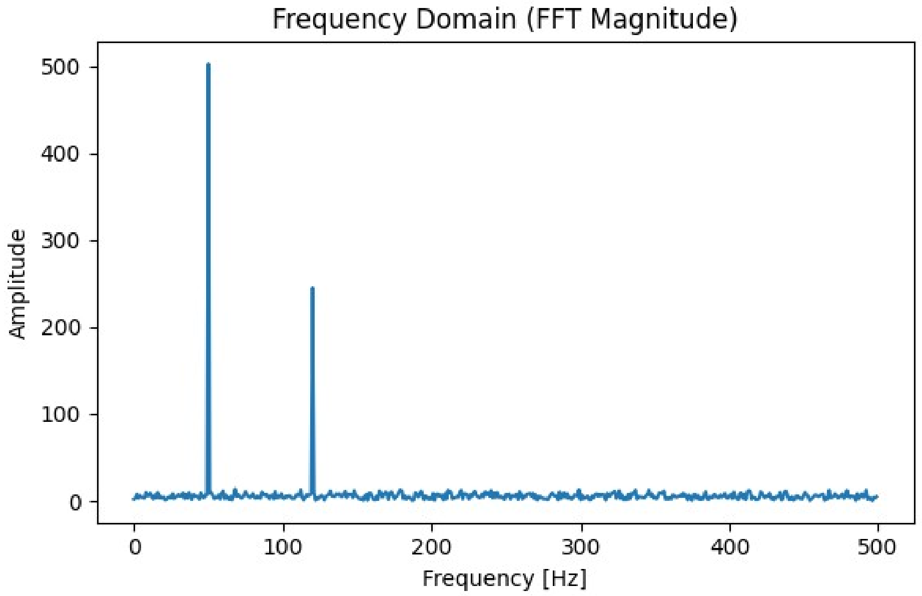
- Time Domain (Raw Voltage)
- Each sample represents a snapshot of voltage at a specific moment, driven by a clock (e.g., 10 kHz sampling rate).
- The waveform shows how voltage varies over time, capturing the analog signal as a series of digital values.
- This is useful for observing trends, noise, and transient behavior.
- Frequency Domain (Post-FFT)
- The FFT transforms the time-domain signal into its frequency components — showing how much of each frequency is present.
- The sharp peak in the spectrum indicates a dominant frequency — likely a periodic component in the original signal.
- The rapid decline after the peak suggests that most of the signal’s energy is concentrated in a narrow band, with little high-frequency content.
- The flat baseline elsewhere means there's minimal noise or harmonic distortion — a clean signal.

Fast Fourier Transform



Simulation


Time Domain (Voltage vs Time)
Voltage
Time [s]
2.0
1.5
1.0
0.5
0.0
−0.5
−1.0
−1.5
−2.0
0.0
0.2
0.4
0.6
0.8
1.0


Frequency Domain (FFT Magnitude)
Amplitude
Frequency [Hz]
500
400
300
200
100
0
0
100
200
300
400
500

Bibliography for FFT

- Horowitz, P., & Hill, W. (2015). The Art of Electronics (3rd ed.). Cambridge University Press.
- Sedgewick, R. (1992). Algorithms in C++. Addison-Wesley.
- Cormen, T. H., Leiserson, C. E., Rivest, R. L., & Stein, C. (2009). Introduction to Algorithms (3rd ed.). MIT Press

GPU Autotuning & Monitoring

- - Setting GPU to autotune mode
- - Using NVIDIA Performance App for live metrics
- - Complementary logging via `monitor.py`
-

Library Setup: CUDA vs MKL

- - Dual-path FFT implementation
- - Comparison goals: throughput, latency, resource usage
- - Performance tradeoffs between CUDA and MKL
- - At small chunk sizes: - MKL benefits from tight CPU cache locality and low overhead - CUDA suffers from kernel launch latency and PCIe transfer costs
- - As chunk size increases: - CUDA amortizes launch and transfer overhead across more data - GPU parallelism becomes dominant, especially for batched FFTs - MKL hits memory bandwidth and thread contention limits -
- Result: - CUDA overtakes MKL in throughput and per-float latency beyond a certain chunk threshold - Especially true when host-device marshalling is optimized and telemetry is decoupled
-

Build Configuration

- - main.cu: hardcoded paths for includes, libs
- - Required setup:
-   - #include paths
-   - libpath and libs
-   - LOG_CUDA, LOG_MKL, LOG_TELEMETRY

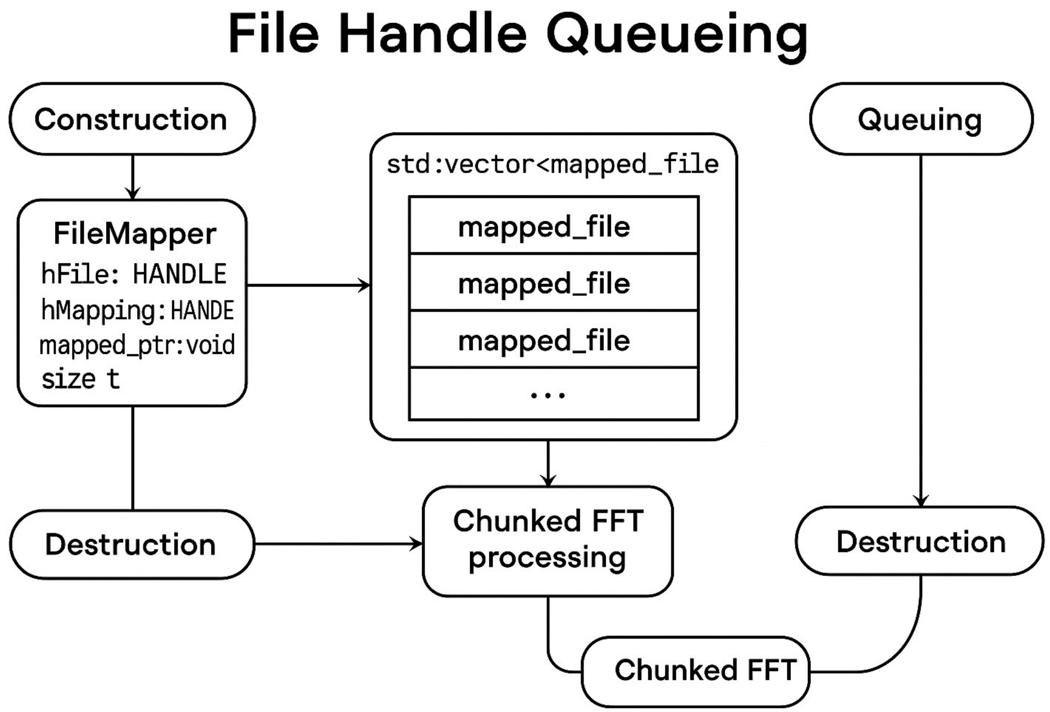
Telemetry Decoupling

- - Why telemetry logging is separated from performance benchmarking
- - Impact on latency and GPU utilization
- - Diagram showing decoupled data paths

File Handle Queuing

- - Strategy for preloading and queuing file handles
- - Benefits: reduced I/O latency, smoother GPU pipeline
- - FileMapper class, std::vector of mapped files.

File Handle Queue and Lifetime Management



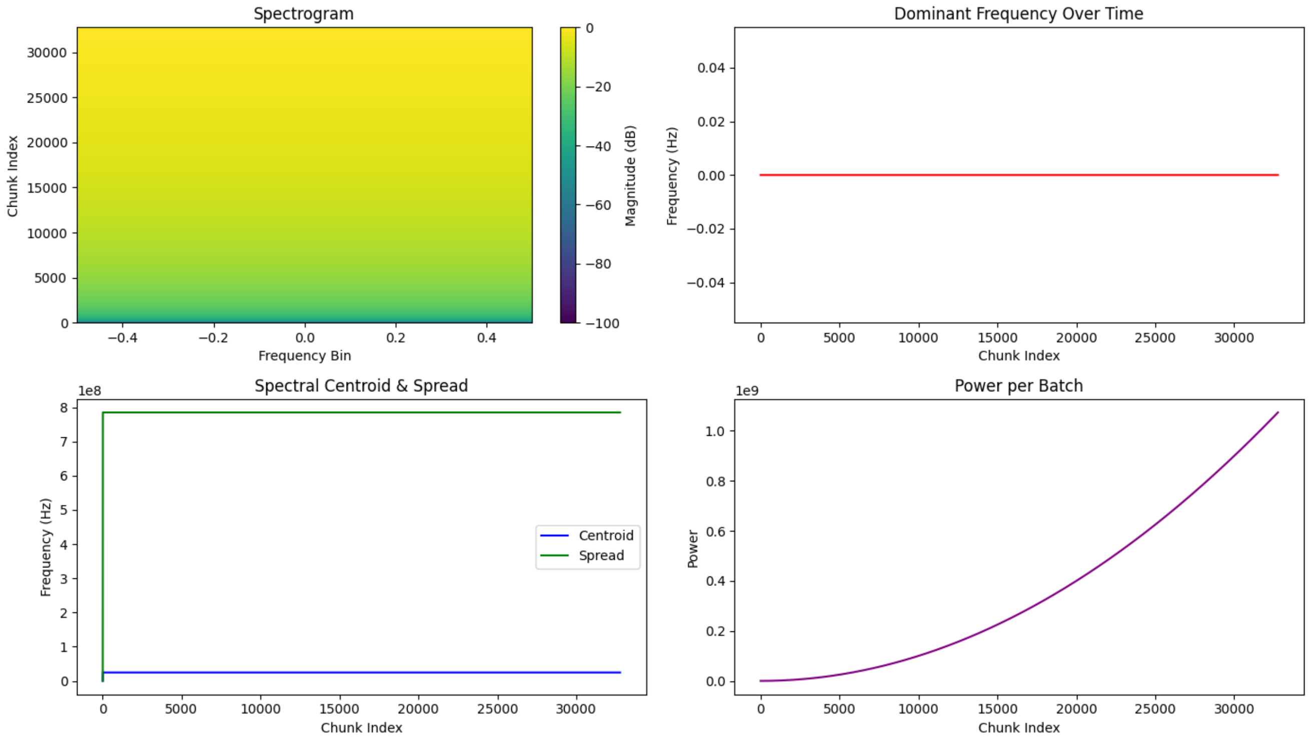
Execution Workflow

- - Step-by-step:
- - Start `monitor.py`
- - Launch `CudaBigData.exe`
- - Run `benchmark.py`
- - If telemetry enabled: run `analyze_spectrum_report.py`
- - Terminal screenshots or command sequence

Report Interpretation: Spectrum Analysis

- - analyze_spectrum_report.py output
- - Axis definitions:
-   - X-axis: frequency bins
-   - Y-axis: amplitude or power
- - Variables:
-   - Sampling rate
-   - FFT size
-   - Windowing function (if any)
- - Annotated spectrum plot
-

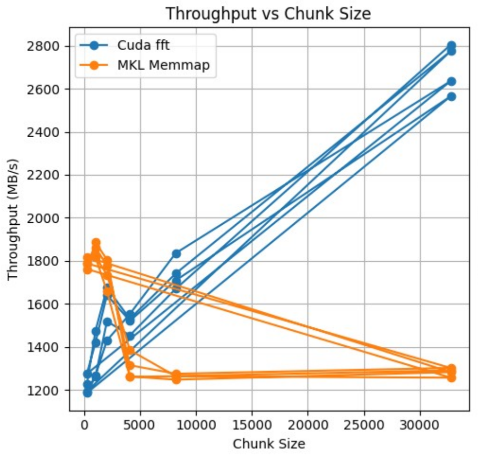
Spectrum Analysis

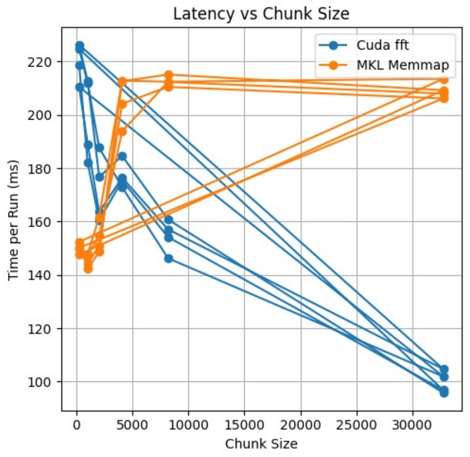


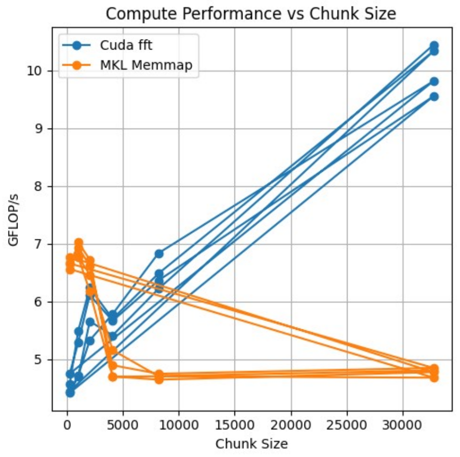
Report Interpretation: Benchmark Metrics

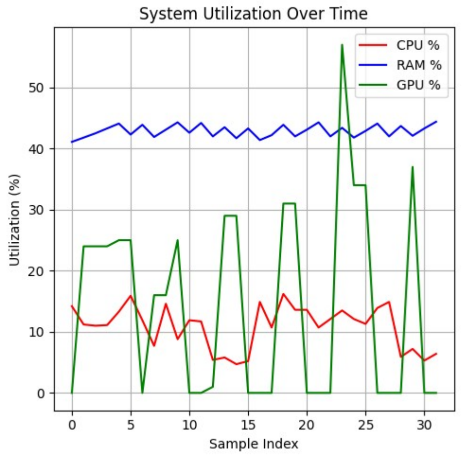
- - `benchmark.py` output
- - Metrics:
-   - Per-float latency
-   - Chunk throughput
-   - GPU vs CPU comparison
- - Bar chart or line graph with callouts
-

Benchmark Diagram









Summary & Takeaways

- Key performance insights
- - Lessons learned from telemetry decoupling
- - Next steps: tuning, scaling, open-source release
- - Host-device marshalling: performance degradation when attempting in-place interleaving or preprocessing on device; forced host-side staging and copy-back due to lack of direct access to mapped buffers, increasing PCIe traffic and latency
- - Mitigation: batch transfers with pinned memory and deferred preprocessing to reduce contention and improve overlap

Links

- The code:
- https://github.com/stevemac321/cuda_fft_batch_mmap_1g
- My linkedin:
- https://www.linkedin.com/in/stevemac321/
- My Technology and Music YouTube Channel:
- https://www.youtube.com/@stephenmackenzie6782